

Programming Languages and Machine Learning

Martin Vechev

DeepCode.ai and ETH Zurich

PL Research: Last 10 years (sample)

- (Semi-) Automated Program Synthesis
 - Mostly learning functions/algorithms over discrete spaces
(from examples, natural language, components, partial specs, etc)
- Automated Symbolic Reasoning
 - Abstract Interpretation = theory of sound & precise approximation
 - SMT solvers
- Approximate/Probabilistic Programming
 - Applications/Analysis/Synthesis

Two part talk (22 + 3) 😊

Learning-based Programming Engines

SLANG

JS NICE

NICE 2 Predict

DEGUARD

Deep3

1. **Pick** a structure of interest, e.g., trees:



2. **Define** a DSL for expressing functions:
(can be Turing complete)

```
TCond ::= z | WriteOp TCond | MoveOp TCond
MoveOp ::= Up, Left, Right, DownFirst, DownLast,
           NextDFS, PrevDFS, NextLeaf, PrevLeaf,
           PrevNodeType, PrevNodeValue,
WriteOp ::= WriteValue, WriteType, WritePos
```

3. **Synthesize** $f_{best} \in \text{DSL}$ from Dataset D:

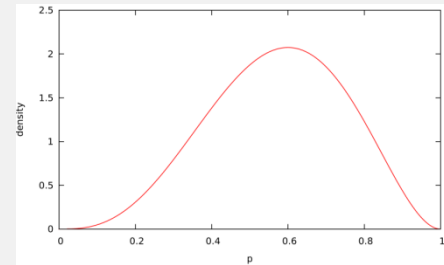
$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}} \text{cost}(D, f)$$

4. **Use** f_{best} on new structures:

$$f_{best} \left(\begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \end{array} \right) \rightarrow \gamma$$

PSI: Exact Solver for Probabilistic Programs

```
def main() {
  p := Uniform(0,1);
  r := [1,1,0,1,0];
  for i in [0..r.len] {
    observe(
      Bernoulli(p) == r[i]);
  }
  return p;
}
```



<http://plml.ethz.ch>

<http://psisolver.org>

Two part talk (22 + 3) 😊

Learning-based Programming Engines

SLANG

JS NICE

NICE 2 Predict

DE&UARD

Deep3

1. **Pick** a structure of interest, e.g., trees:



2. **Define** a DSL for expressing functions:
(can be Turing complete)

```
TCond ::= z | WriteOp TCond | MoveOp TCond
MoveOp ::= Up, Left, Right, DownFirst, DownLast,
NextDFS, PrevDFS, NextLeaf, PrevLeaf,
PrevNodeType, PrevNodeValue,
WriteOp ::= WriteValue, WriteType, WritePos
```

3. **Synthesize** $f_{best} \in \text{DSL}$ from Dataset D:

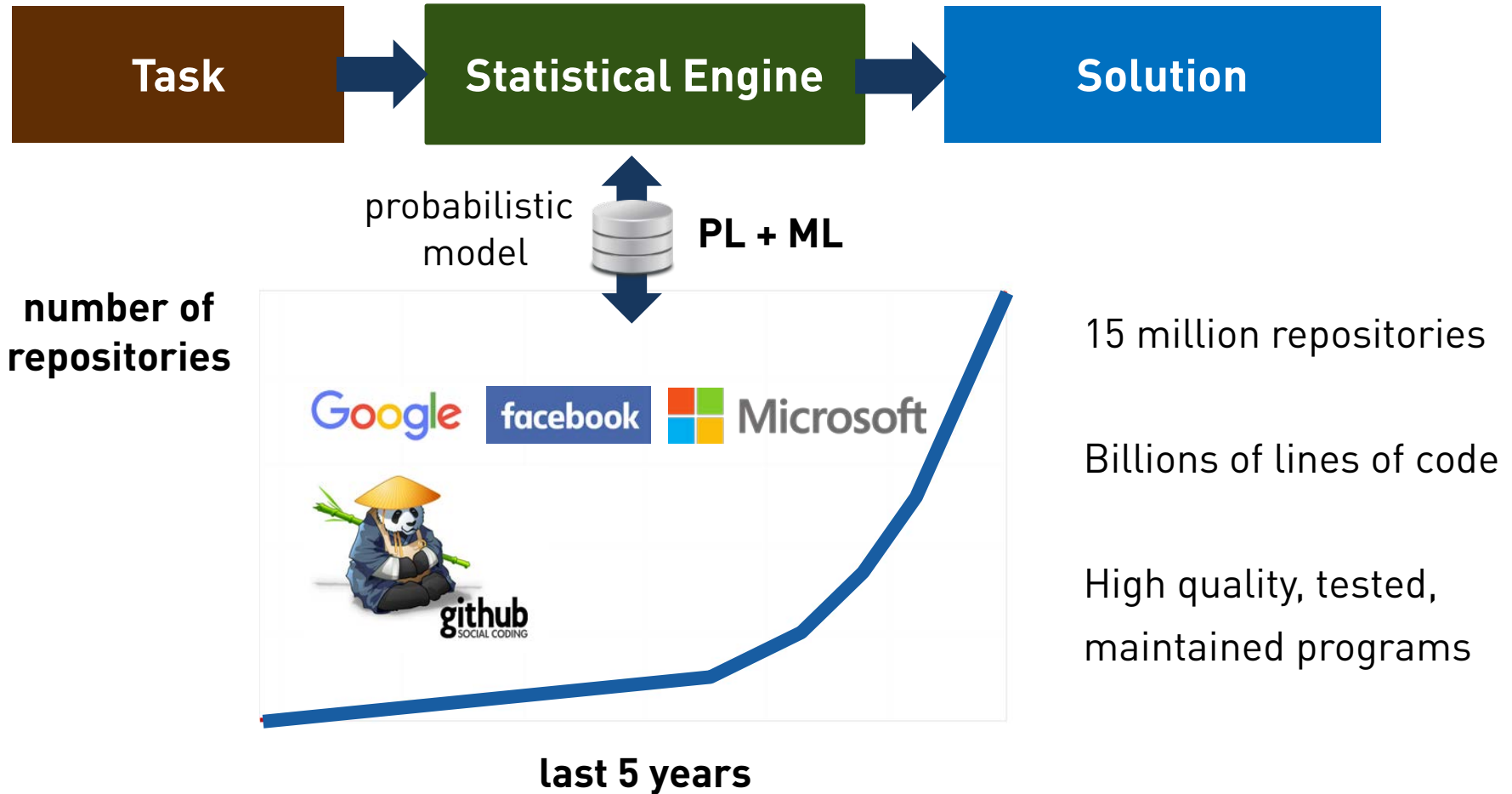
$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}} \text{cost}(D, f)$$

4. **Use** f_{best} on new structures:

$$f_{best} \left(\begin{array}{c} \diagup \\ \diagdown \end{array} \right) \rightarrow \gamma$$

<http://plml.ethz.ch>

Probabilistic Learning from Code



Probabilistic Learning from Code

Probabilistically likely solutions to problems hard to solve otherwise

Joint work with :



Veselin
Raychev



Andreas
Krause



Pavol
Bielik



Christine
Zeller



Svetoslav
Karaivanov



Pascal
Roos



Benjamin
Bischel



Timon
Gehr



Petar
Tsankov



Mateo
Panzacchi

Publications

- Program Synthesis for Char. Level Language Modeling, **ICLR'17 sub**
- Learning a Static Analyzer from Data, <https://arxiv.org/abs/1611.01752>
- Statistical Deobfuscation of Android Applications, **ACM CCS'16**
- Probabilistic Mode for Code with Decision Trees, **ACM OOPSLA'16**
- PHOG: Probabilistic Mode for Code, **ACM ICML'16**
- Learning Programs from Noisy Data, **ACM POPL'16**
- Predicting Program Properties from "Big Code", **ACM POPL'15**
- Code Completion with Statistical Language Models, **ACM PLDI'14**
- Machine Translation for Programming Languages, **ACM Onward'14**

Statistical Engines

apk-deguard.com



jsnice.org



nice2predict.org



more: <http://plml.ethz.ch>

A Key Question

Data

Learning

Model



Probabilistic
Model



Widely
Applicable

Efficient
Learning

High
Precision

Explainable
Predictions

Training dataset ***D***

```
f.open("f2" | "r");  
f.read();
```

```
f.open("f2" | "w");  
f.write("c");
```

```
f.open("f1" | "r");  
f.read();
```

query:

```
f.open("file" | "r");  
f. ?
```

Training dataset D

```
f.open("f2" | "r");  
f.read();
```

```
f.open("f2" | "w");  
f.write("c");
```

```
f.open("f1" | "r");  
f.read();
```

query:

```
f.open("file" | "r");  
f. ?
```

3-gram model on tokens

Hindle et. al., ACM ICSE'12

P(open		f.)	~ 3/6
P(read		f.)	~ 2/6
P(write		f.)	~ 1/6


context γ

Training dataset D

```
f.open("f2" | "r");  
f.read();
```

```
f.open("f2" | "w");  
f.write("c");
```

```
f.open("f1" | "r");  
f.read();
```

query:

```
f.open("file" | "r");  
f. open
```

3-gram model on tokens

Hindle et. al., ACM ICSE'12

P(open		f.)	~ 3/6
P(read		f.)	~ 2/6
P(write		f.)	~ 1/6


context γ

Training dataset D

```
f.open("f2" | "r");  
f.read();
```

```
f.open("f2" | "w");  
f.write("c");
```

```
f.open("f1" | "r");  
f.read();
```

probabilistic model on APIs

Raychev et. al., ACM PLDI'14

$P(\text{read} \mid \text{open})$	$\sim 2/3$
$P(\text{write} \mid \text{open})$	$\sim 1/3$

context γ

query:

```
f.open("file" | "r");  
f. ?
```

3-gram model on tokens

Hindle et. al., ACM ICSE'12

$P(\text{open} \mid \text{f.})$	$\sim 3/6$
$P(\text{read} \mid \text{f.})$	$\sim 2/6$
$P(\text{write} \mid \text{f.})$	$\sim 1/6$

context γ

Training dataset D

```
f.open("f2" | "r");  
f.read();
```

```
f.open("f2" | "w");  
f.write("c");
```

```
f.open("f1" | "r");  
f.read();
```

probabilistic model on APIs

Raychev et. al., ACM PLDI'14

$P(\text{read} \mid$	$\text{open}) \sim 2/3$
$P(\text{write} \mid$	$\text{open}) \sim 1/3$

context γ

query:

```
f.open("file" | "r");  
f.read
```

3-gram model on tokens

Hindle et. al., ACM ICSE'12

$P(\text{open} \mid$	$f.) \sim 3/6$
$P(\text{read} \mid$	$f.) \sim 2/6$
$P(\text{write} \mid$	$f.) \sim 1/6$

context γ

Training dataset D

```
f.open("f2" | "r");  
f.read();
```

```
f.open("f2" | "w");  
f.write("c");
```

```
f.open("f1" | "r");  
f.read();
```

probabilistic model on APIs

Raychev et. al., ACM PLDI'14

$P(\text{read} \mid \text{open})$	$\sim 2/3$
$P(\text{write} \mid \text{open})$	$\sim 1/3$

context γ

query:

```
f.open("file" | "r");  
f. ?
```

3-gram model on tokens

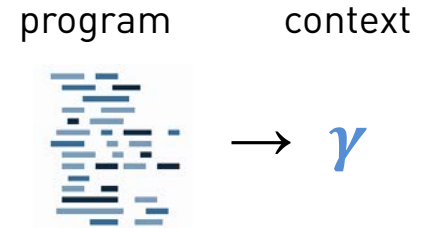
Hindle et. al., ACM ICSE'12

$P(\text{open} \mid \text{f.})$	$\sim 3/6$
$P(\text{read} \mid \text{f.})$	$\sim 2/6$
$P(\text{write} \mid \text{f.})$	$\sim 1/6$

context γ

What should the context be?

key idea: **synthesize a function** f :



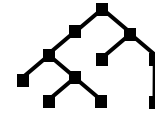
“...All problems in computer science can be solved
by **another level of indirection...**”

-- David Wheeler

Creating probabilistic models: our method

[“Learning Programs from Noisy Data”, ACM POPL’16,
“PHOG: Probabilistic Model for Code”, ACM ICML’16,
“Probabilistic Model for Code with Decision Trees”, ACM OOPSLA’16]

1. Pick a structure of interest, e.g., ASTs:



2. Define a DSL for expressing functions:
(can be Turing complete)

DSL

3. Synthesize $f_{best} \in \text{DSL}$ from Dataset \mathbf{D} :

$$f_{best} = \underset{f \in \text{DSL}}{\operatorname{argmin}} \operatorname{cost}(\mathbf{D}, f)$$

4. Use f_{best} to compute context and predict:

$$f_{best} \left(\text{AST} \right) \rightarrow \gamma$$

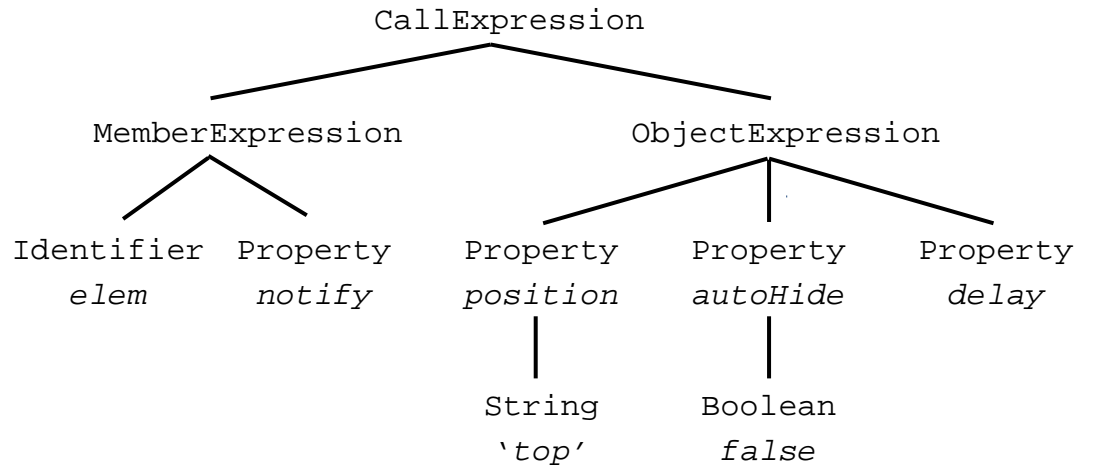
Step 1: Pick Structure of Interest

Let it be abstract syntax trees (ASTs) of programs

JavaScript program

```
elem.notify({  
  position: 'top',  
  autoHide: false,  
  delay: 100  
});
```

AST



Step 2: Define a DSL over structure

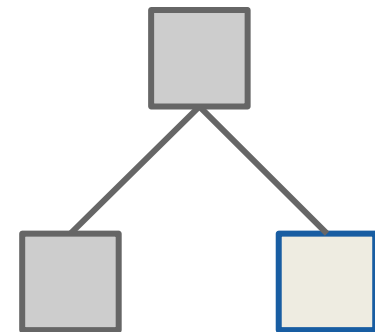
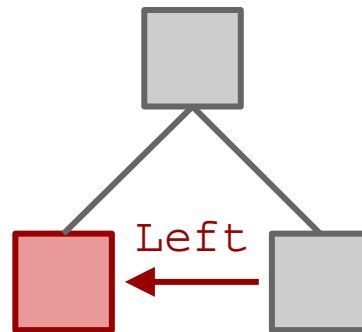
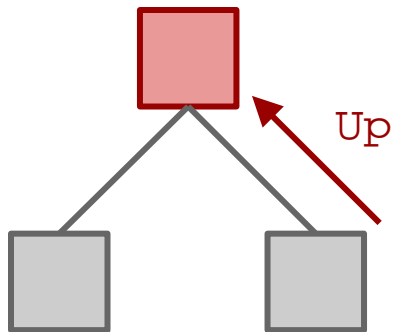
Syntax

TCond ::= ε | WriteOp TCond | MoveOp TCond

MoveOp ::= Up, Left, Right, DownFirst, DownLast,
NextDFS, PrevDFS, NextLeaf,
PrevLeaf, PrevNodeType, PrevNodeValue, PrevNodeContext

WriteOp ::= WriteValue, WriteType, WritePos

Semantics



WriteValue
 $\gamma \leftarrow \gamma \cdot \square$

Step 3: synthesize f_{best}

$$f_{best} = \underset{f \in \text{DSL}}{\operatorname{argmin}} \operatorname{cost}(\mathbf{D}, f)$$

Step 3: synthesize f_{best}

DSL

```
TCond ::= ε | WriteOp TCond | MoveOp TCond
MoveOp ::= Up, Left, Right, ...
WriteOp ::= WriteValue, WriteType, ...
```

generate candidate f

dataset D



millions ($\approx 10^8$)

Synthesizer

$$f_{best} = \operatorname{argmin}_{f \in \text{DSL}} \operatorname{cost}(D, f)$$

Build Probabilistic Model P

use D and f to compute

$$P(\text{element} \mid f(\text{tree}))$$

to scale: iterative synthesis on fraction of examples

$$\operatorname{cost}(D, f) = \operatorname{entropy}(P)$$

$$O(|D|)$$

Step 4: use f_{best} to predict

program

```
elem.notify{  
  ... ,  
  ... ,  
  {  
    position: 'top',  
    hide: false,  
    ?  
  }  
};
```

f_{best}

Left
WriteValue
Up
WritePos
Up
DownFirst
DownLast
WriteValue

Context γ

{ }
{hide}
{hide}
{hide, 3}
{hide, 3}
{hide, 3}
{hide, 3}
{hide, 3, notify}



{Previous Property, Parameter Position, API name}

Deep3: Experimental Results

[Probabilistic Model of JavaScript]

Dataset **D**: 150,000 files Training Time: ~**100** hours $f_{best} \sim$ **50,000** instr.

Probabilistic Model

Accuracy (APIs)

Last two tokens, Hindle et. al. [ICSE'12]

22.2%

Last two APIs, Raychev et. al. [PLDI'14]

30.4%

Deep3

66.6%

Details in: "Probabilistic Model for Code with Decision Trees", ACM OOPSLA'16

Deep3: Experimental Results

[Probabilistic Model of Python]

Dataset **D**: 150,000 files Training Time: ~**100** hours $f_{best} \sim$ **120,000** instr

Probabilistic Model	Accuracy (identifiers)
Last two tokens, Hindle et. al. [ICSE'12]	38%
<i>Deep3</i>	51%

Details in: "Probabilistic Model for Code with Decision Trees", ACM OOPSLA'16

Applying the Concept to Natural Language

[Program Synthesis for Character Level Language Modeling, **ICLR'17 sub**]

Dataset **D**:

Hutter Prize Wikipedia Dataset

Training Time: ~ **8** hours

f_{best} ~ **9,000** instr

uses a char-level DSL with state

Interpretable model, browse here:

<http://www.srl.inf.ethz.ch/charmmodel.html>

Probabilistic Model

Bits-per-Character

7-gram (best)	1.94
Stacked LSTM (Graves 2013)	1.67
Char-based DSL synthesis	1.62
MRNN (Sutskever 2011)	1.60
MI-LSTM (Wu et al. 2016)	1.44
HM-LSTM* (Chung et al. 2016)	1.40

Learning (Abstract) Semantics

[Learning a Static Analyzer from Data, <https://arxiv.org/abs/1611.01752>]

```
function isBig(v) {  
  return v < this.length  
}  
[12, 5].filter(isBig);
```

```
VarPtsTo("global", h)  
checkIfInsideMethodCall  
checkMethodCallName  
checkReceiverType  
checkNumberOfArguments ...
```

```
VarPtsTo(this, h)
```

Can be understood by experts

Found issues in Facebook's Flow



DEEPCODE

<http://deepcode.ai>

ETH spin-off, co-founded in 2016 by

From code to predictions

Handles any programming language



Veselin
Raychev



Martin
Vechev

We are hiring!

More Info

**Learning from Large Codebases,
PhD Thesis, ETH Zurich, 2016**



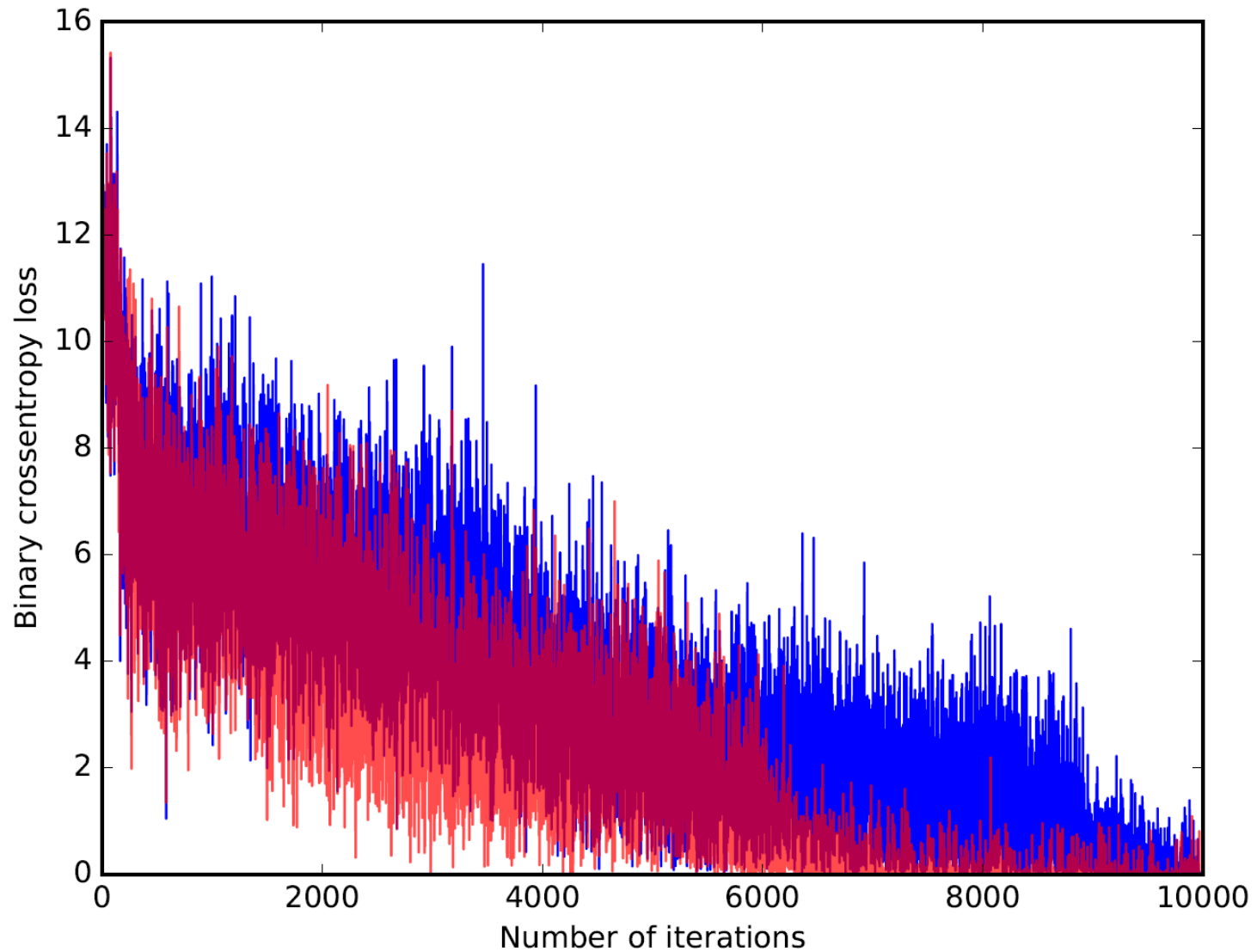
Veselin
Raychev

<http://plml.ethz.ch>

Dagstuhl Seminar on Big Code Analytics, Nov 2015

Data sets, tools, challenges: <http://learningfrombigcode.org>

Synthesis with NTMs



Two part talk (22 + 3) 😊

Learning-based Programming Engines

SLANG

JS NICE

NICE 2 Predict

DE&UARD

Deep3

1. **Pick** a structure of interest, e.g., trees:



2. **Define** a DSL for expressing functions:
(can be Turing complete)

```
TCond ::= z | WriteOp TCond | MoveOp TCond
MoveOp ::= Up, Left, Right, DownFirst, DownLast,
           NextDFS, PrevDFS, NextLeaf, PrevLeaf,
           PrevNodeType, PrevNodeValue,
WriteOp ::= WriteValue, WriteType, WritePos
```

3. **Synthesize** $f_{best} \in \text{DSL}$ from Dataset D:

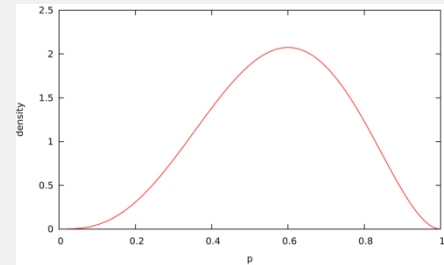
$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}} \text{cost}(D, f)$$

4. **Use** f_{best} on new structures:

$$f_{best} \left(\begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \end{array} \right) \rightarrow \gamma$$

PSI: Exact Solver for Probabilistic Programs

```
def main() {
  p := Uniform(0,1);
  r := [1,1,0,1,0];
  for i in [0..r.len] {
    observe(
      Bernoulli(p) == r[i]);
  }
  return p;
}
```



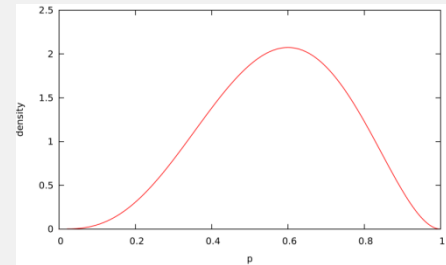
<http://plml.ethz.ch>

<http://psisolver.org>

Two part talk (22 + 3) 😊

PSI: Exact Solver for Probabilistic Programs

```
def main() {  
  p := Uniform(0,1);  
  r := [1,1,0,1,0];  
  for i in [0..r.len] {  
    observe(  
      Bernoulli(p) == r[i]);  
    }  
  return p;  
}
```



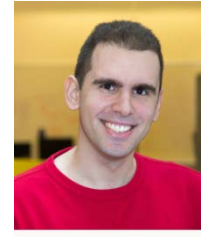
<http://psisolver.org>

PSI

PSI: Exact Symbolic Inference for Probabilistic Programs, CAV'16



Timon
Gehr



Sasa
Misailovic

Exact inference for probabilistic programs

<http://psisolver.org/>

PSI

```
def max(a,b) {
  r := a;
  if b > r { r = b; }
  return r;
}

def main() {
  x := Uniform(0,1);
  y := Gauss(0,1);
  z := Uniform(0,1);
  r := max(x,max(y,z));
  observe(x < 0.75);
  if Bernoulli(1/2)
    { assert(r < 0.9); }
  return r + UniformInt(1,3);
}
```


PSI

```
def max(a,b) {
```

```
  r := a;
```

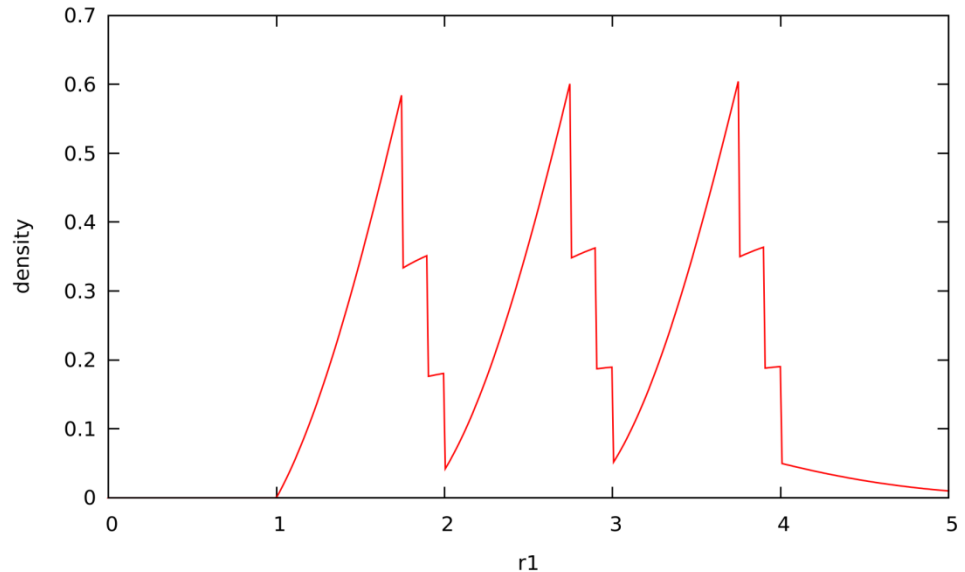
```
  if r <= 0 {  
    r := 0;  
  }  
  if r > 1 {  
    r := 1;  
  }  
  if r < 0.5 {  
    r := 0.5;  
  }  
  if r > 0.5 {  
    r := 1 - r;  
  }  
  if r < 0.25 {  
    r := 0.25;  
  }  
  if r > 0.75 {  
    r := 0.75;  
  }  
  if r < 0.125 {  
    r := 0.125;  
  }  
  if r > 0.875 {  
    r := 0.875;  
  }  
  if r < 0.0625 {  
    r := 0.0625;  
  }  
  if r > 0.9375 {  
    r := 0.9375;  
  }  
  if r < 0.03125 {  
    r := 0.03125;  
  }  
  if r > 0.96875 {  
    r := 0.96875;  
  }  
  if r < 0.015625 {  
    r := 0.015625;  
  }  
  if r > 0.984375 {  
    r := 0.984375;  
  }  
  if r < 0.0078125 {  
    r := 0.0078125;  
  }  
  if r > 0.9921875 {  
    r := 0.9921875;  
  }  
  if r < 0.00390625 {  
    r := 0.00390625;  
  }  
  if r > 0.99609375 {  
    r := 0.99609375;  
  }  
  if r < 0.001953125 {  
    r := 0.001953125;  
  }  
  if r > 0.998046875 {  
    r := 0.998046875;  
  }  
  if r < 0.0009765625 {  
    r := 0.0009765625;  
  }  
  if r > 0.9990234375 {  
    r := 0.9990234375;  
  }  
  if r < 0.00048828125 {  
    r := 0.00048828125;  
  }  
  if r > 0.99951171875 {  
    r := 0.99951171875;  
  }  
  if r < 0.000244140625 {  
    r := 0.000244140625;  
  }  
  if r > 0.999755859375 {  
    r := 0.999755859375;  
  }  
  if r < 0.0001220703125 {  
    r := 0.0001220703125;  
  }  
  if r > 0.9998779296875 {  
    r := 0.9998779296875;  
  }  
  if r < 6.103515625e-05 {  
    r := 6.103515625e-05;  
  }  
  if r > 0.99993896484375 {  
    r := 0.99993896484375;  
  }  
  if r < 3.0517578125e-05 {  
    r := 3.0517578125e-05;  
  }  
  if r > 0.999969482421875 {  
    r := 0.999969482421875;  
  }  
  if r < 1.52587890625e-05 {  
    r := 1.52587890625e-05;  
  }  
  if r > 0.9999847412109375 {  
    r := 0.9999847412109375;  
  }  
  if r < 7.62939453125e-06 {  
    r := 7.62939453125e-06;  
  }  
  if r > 0.99999237060546875 {  
    r := 0.99999237060546875;  
  }  
  if r < 3.814697265625e-06 {  
    r := 3.814697265625e-06;  
  }  
  if r > 0.999996185302734375 {  
    r := 0.999996185302734375;  
  }  
  if r < 1.9073486328125e-06 {  
    r := 1.9073486328125e-06;  
  }  
  if r > 0.9999980926513671875 {  
    r := 0.9999980926513671875;  
  }  
  if r < 9.5367431640625e-07 {  
    r := 9.5367431640625e-07;  
  }  
  if r > 0.99999904632568359375 {  
    r := 0.99999904632568359375;  
  }  
  if r < 4.76837158203125e-07 {  
    r := 4.76837158203125e-07;  
  }  
  if r > 0.999999523162841796875 {  
    r := 0.999999523162841796875;  
  }  
  if r < 2.384185791015625e-07 {  
    r := 2.384185791015625e-07;  
  }  
  if r > 0.9999997615814208984375 {  
    r := 0.9999997615814208984375;  
  }  
  if r < 1.1920928955078125e-07 {  
    r := 1.1920928955078125e-07;  
  }  
  if r > 0.99999988079071044921875 {  
    r := 0.99999988079071044921875;  
  }  
  if r < 5.9604644775390625e-08 {  
    r := 5.9604644775390625e-08;  
  }  
  if r > 0.999999940395355224609375 {  
    r := 0.999999940395355224609375;  
  }  
  if r < 2.98023223876953125e-08 {  
    r := 2.98023223876953125e-08;  
  }  
  if r > 0.9999999701976776123046875 {  
    r := 0.9999999701976776123046875;  
  }  
  if r < 1.490116119384765625e-08 {  
    r := 1.490116119384765625e-08;  
  }  
  if r > 0.999999985098838806171875 {  
    r := 0.999999985098838806171875;  
  }  
  if r < 7.450580596923828125e-09 {  
    r := 7.450580596923828125e-09;  
  }  
  if r > 0.99999999254941940309375 {  
    r := 0.99999999254941940309375;  
  }  
  if r < 3.7252902984619140625e-09 {  
    r := 3.7252902984619140625e-09;  
  }  
  if r > 0.999999996274709701546875 {  
    r := 0.999999996274709701546875;  
  }  
  if r < 1.86264514923095703125e-09 {  
    r := 1.86264514923095703125e-09;  
  }  
  if r > 0.999999998137354850771875 {  
    r := 0.999999998137354850771875;  
  }  
  if r < 9.31322574615478515625e-10 {  
    r := 9.31322574615478515625e-10;  
  }  
  if r > 0.9999999990686774253859375 {  
    r := 0.9999999990686774253859375;  
  }  
  if r < 4.656612873077392578125e-10 {  
    r := 4.656612873077392578125e-10;  
  }  
  if r > 0.99999999953433871269296875 {  
    r := 0.99999999953433871269296875;  
  }  
  if r < 2.3283064365386962890625e-10 {  
    r := 2.3283064365386962890625e-10;  
  }  
  if r > 0.999999999767169356346484375 {  
    r := 0.999999999767169356346484375;  
  }  
  if r < 1.16415321826934814453125e-10 {  
    r := 1.16415321826934814453125e-10;  
  }  
  if r > 0.9999999998835846781732421875 {  
    r := 0.9999999998835846781732421875;  
  }  
  if r < 5.82076609134674072265625e-11 {  
    r := 5.82076609134674072265625e-11;  
  }  
  if r > 0.99999999994179233908662109375 {  
    r := 0.99999999994179233908662109375;  
  }  
  if r < 2.9103830456733703611328125e-11 {  
    r := 2.9103830456733703611328125e-11;  
  }  
  if r > 0.99999999997089616954331046875 {  
    r := 0.99999999997089616954331046875;  
  }  
  if r < 1.45519152283668518056640625e-11 {  
    r := 1.45519152283668518056640625e-11;  
  }  
  if r > 0.999999999985448084771655234375 {  
    r := 0.999999999985448084771655234375;  
  }  
  if r < 7.27595761418342590283203125e-12 {  
    r := 7.27595761418342590283203125e-12;  
  }  
  if r > 0.9999999999927240423858276171875 {  
    r := 0.9999999999927240423858276171875;  
  }  
  if r < 3.637978807091712951416015625e-12 {  
    r := 3.637978807091712951416015625e-12;  
  }  
  if r > 0.99999999999636202119291380859375 {  
    r := 0.99999999999636202119291380859375;  
  }  
  if r < 1.81898940354585647570640625e-12 {  
    r := 1.81898940354585647570640625e-12;  
  }  
  if r > 0.9999999999981810105964569296875 {  
    r := 0.9999999999981810105964569296875;  
  }  
  if r < 9.09494701772928237853203125e-13 {  
    r := 9.09494701772928237853203125e-13;  
  }  
  if r > 0.999999999999090505298228484375 {  
    r := 0.999999999999090505298228484375;  
  }  
  if r < 4.547473508864641189266015625e-13 {  
    r := 4.547473508864641189266015625e-13;  
  }  
  if r > 0.9999999999995452526491142421875 {  
    r := 0.9999999999995452526491142421875;  
  }  
  if r < 2.2737367544323205946330078125e-13 {  
    r := 2.2737367544323205946330078125e-13;  
  }  
  if r > 0.99999999999977262632455712109375 {  
    r := 0.99999999999977262632455712109375;  
  }  
  if r < 1.13686837721616029731650390625e-13 {  
    r := 1.13686837721616029731650390625e-13;  
  }  
  if r > 0.99999999999988631316227783546875 {  
    r := 0.99999999999988631316227783546875;  
  }  
  if r < 5.68434188608080148658251953125e-14 {  
    r := 5.68434188608080148658251953125e-14;  
  }  
  if r > 0.99999999999994315658113891771875 {  
    r := 0.99999999999994315658113891771875;  
  }  
  if r < 2.842170943040400743291259765625e-14 {  
    r := 2.842170943040400743291259765625e-14;  
  }  
  if r > 0.99999999999997157829056945884375 {  
    r := 0.99999999999997157829056945884375;  
  }  
  if r < 1.4210854715202003716456298828125e-14 {  
    r := 1.4210854715202003716456298828125e-14;  
  }  
  if r > 0.999999999999985794145284729421875 {  
    r := 0.999999999999985794145284729421875;  
  }  
  if r < 7.1054273576010018582281494140625e-15 {  
    r := 7.1054273576010018582281494140625e-15;  
  }  
  if r > 0.9999999999999928970726423647109375 {  
    r := 0.9999999999999928970726423647109375;  
  }  
  if r < 3.55271367880050092911407470625e-15 {  
    r := 3.55271367880050092911407470625e-15;  
  }  
  if r > 0.9999999999999964485363211823546875 {  
    r := 0.9999999999999964485363211823546875;  
  }  
  if r < 1.776356839400250464557037353125e-15 {  
    r := 1.776356839400250464557037353125e-15;  
  }  
  if r > 0.99999999999999822426816059117734375 {  
    r := 0.99999999999999822426816059117734375;  
  }  
  if r < 8.881784197001252322785186765625e-16 {  
    r := 8.881784197001252322785186765625e-16;  
  }  
  if r > 0.999999999999999112134080295588671875 {  
    r := 0.999999999999999112134080295588671875;  
  }  
  if r < 4.44089209850062616139259337828125e-16 {  
    r := 4.44089209850062616139259337828125e-16;  
  }  
  if r > 0.99999999999999955607104014779434375 {  
    r := 0.99999999999999955607104014779434375;  
  }  
  if r < 2.220446049250313080696296689140625e-16 {  
    r := 2.220446049250313080696296689140625e-16;  
  }  
  if r > 0.999999999999999778035520073897171875 {  
    r := 0.999999999999999778035520073897171875;  
  }  
  if r < 1.11022302462515654034814834453125e-16 {  
    r := 1.11022302462515654034814834453125e-16;  
  }  
  if r > 0.9999999999999998890177600369485859375 {  
    r := 0.9999999999999998890177600369485859375;  
  }  
  if r < 5.55111512312578270174074172265625e-17 {  
    r := 5.55111512312578270174074172265625e-17;  
  }  
  if r > 0.99999999999999994450888001847429296875 {  
    r := 0.99999999999999994450888001847429296875;  
  }  
  if r < 2.775557561562891350870370861328125e-17 {  
    r := 2.775557561562891350870370861328125e-17;  
  }  
  if r > 0.999999999999999972252880009237146484375 {  
    r := 0.999999999999999972252880009237146484375;  
  }  
  if r < 1.3877787807814456754351854306640625e-17 {  
    r := 1.3877787807814456754351854306640625e-17;  
  }  
  if r > 0.9999999999999999861264400046185732421875 {  
    r := 0.9999999999999999861264400046185732421875;  
  }  
  if r < 6.9388939039072283771759271533203125e-18 {  
    r := 6.9388939039072283771759271533203125e-18;  
  }  
  if r > 0.99999999999999999306322000230928662109375 {  
    r := 0.99999999999999999306322000230928662109375;  
  }  
  if r < 3.46944695195361418858796357666015625e-18 {  
    r := 3.46944695195361418858796357666015625e-18;  
  }  
  if r > 0.99999999999999999653161000115464331046875 {  
    r := 0.99999999999999999653161000115464331046875;  
  }  
  if r < 1.734723475976807094293981788330078125e-18 {  
    r := 1.734723475976807094293981788330078125e-18;  
  }  
  if r > 0.999999999999999998265805000577321655234375 {  
    r := 0.999999999999999998265805000577321655234375;  
  }  
  if r < 8.67361737988403547146990894166515625e-19 {  
    r := 8.67361737988403547146990894166515625e-19;  
  }  
  if r > 0.9999999999999999991329025002886608276171875 {  
    r := 0.9999999999999999991329025002886608276171875;  
  }  
  if r < 4.336808689942017735734954470832578125e-19 {  
    r := 4.336808689942017735734954470832578125e-19;  
  }  
  if r > 0.99999999999999999956580125014433041380859375 {  
    r := 0.99999999999999999956580125014433041380859375;  
  }  
  if r < 2.168404344971008867867477235416628125e-19 {  
    r := 2.168404344971008867867477235416628125e-19;  
  }  
  if r > 0.9999999999999999997829006250721652069046875 {  
    r := 0.9999999999999999997829006250721652069046875;  
  }  
  if r < 1.0842021724855044339337386177083125e-19 {  
    r := 1.0842021724855044339337386177083125e-19;  
  }  
  if r > 0.99999999999999999989145031253608260345234375 {  
    r := 0.99999999999999999989145031253608260345234375;  
  }  
  if r < 5.42101086242752216966869308885415625e-20 {  
    r := 5.42101086242752216966869308885415625e-20;  
  }  
  if r > 0.999999999999999999945725156268041301726171875 {  
    r := 0.999999999999999999945725156268041301726171875;  
  }  
  if r < 2.710505431213761084834346544427078125e-20 {  
    r := 2.710505431213761084834346544427078125e-20;  
  }  
  if r > 0.9999999999999999999728625781340206508631046875 {  
    r := 0.9999999999999999999728625781340206508631046875;  
  }  
  if r < 1.3552527156068805424171732722135390625e-20 {  
    r := 1.3552527156068805424171732722135390625e-20;  
  }  
  if r > 0.9999999999999999999864312890670125254171875 {  
    r := 0.9999999999999999999864312890670125254171875;  
  }  
  if r < 6.7762635780344027120858663610676953125e-21 {  
    r := 6.7762635780344027120858663610676953125e-21;  
  }  
  if r > 0.99999999999999999999321564453501251270859375 {  
    r := 0.99999999999999999999321564453501251270859375;  
  }  
  if r < 3.38813178901720135604293318053384765625e-21 {  
    r := 3.38813178901720135604293318053384765625e-21;  
  }  
  if r > 0.99999999999999999999643128906701251270859375 {  
    r := 0.99999999999999999999643128906701251270859375;  
  }  
  if r < 1.694065894508600678021466590266923828125e-21 {  
    r := 1.694065894508600678021466590266923828125e-21;  
  }  
  if r > 0.99999999999999999999821564453501251270859375 {  
    r := 0.99999999999999999999821564453501251270859375;  
  }  
  if r < 8.470329472543003390107332951334619140625e-22 {  
    r := 8.470329472543003390107332951334619140625e-22;  
  }  
  if r > 0.9999999999999999999991078222675062563546875 {  
    r := 0.9999999999999999999991078222675062563546875;  
  }  
  if r < 4.235164736271501695053666475667309375e-22 {  
    r := 4.235164736271501695053666475667309375e-22;  
  }  
  if r > 0.999999999999999999999539111133753128171875 {  
    r := 0.999999999999999999999539111133753128171875;  
  }  
  if r < 2.1175823681357508475268332378336546875e-22 {  
    r := 2.1175823681357508475268332378336546875e-22;  
  }  
  if r > 0.9999999999999999999997695555675062563546875 {  
    r := 0.9999999999999999999997695555675062563546875;  
  }  
  if r < 1.05879118406787542376341661891682734375e-22 {  
    r := 1.05879118406787542376341661891682734375e-22;  
  }  
  if r > 0.999999999999999999999884777783753128171875 {  
    r := 0.999999999999999999999884777783753128171875;  
  }  
  if r < 5.2939559203393771188170833094584140625e-23 {  
    r := 5.2939559203393771188170833094584140625e-23;  
  }  
  if r > 0.999999999999999999999942388891875640859375 {  
    r := 0.999999999999999999999942388891875640859375;  
  }  
  if r < 2.646977960169688559440891654729209375e-23 {  
    r := 2.646977960169688559440891654729209375e-23;
```

PSI

```
def max(a,b) {  
  r := a;  
  if b > r { r = b; }  
  return r;  
}  
  
def main() {  
  x := Uniform(0,1);  
  y := Gauss(0,1);  
  z := Uniform(0,1);  
  r := max(x,max(y,z));  
  observe(x < 0.75);  
  if Bernoulli(1/2)  
    { assert(r < 0.9); }  
  return r + UniformInt(1,3);  
}
```



a little nicer to look at...



can compute various queries on the PDF:
e.g., **expectations**, **marginal probabilities**

$\mathbf{E}[\text{result}] \approx 2.6929$

$\mathbf{Pr}[\text{error}] \approx 0.132827$

PSI: Ingredients

Symbolic Domain for PDFs

$$\begin{aligned}
 e \in E ::= & \quad x \mid e \mid \pi \mid 0 \mid 1 \mid 2 \mid \dots \\
 & \mid \log(e) \mid -e \mid e_1 + \dots + e_n \mid e_1 \cdot \dots \cdot e_n \mid e_1^{e_2} \\
 & \mid \delta(e) \mid [e_1 = e_2] \mid [e_1 \leq e_2] \mid [e_1 \neq e_2] \mid [e_1 < e_2] \\
 & \mid \int_{\mathbb{R}} dx e[[x]] \mid \sum_{x \in \mathbb{Z}} e[[x]] \mid \varphi(e_1, \dots, e_n) \\
 & \mid (d/dx)^{-1}[e^{-x^2}](e) \text{ (Error function)}
 \end{aligned}$$

Encodes probability density functions

- ▶ Bernoulli($x; p$) = $p \cdot \delta(1 - x) + (1 - p) \cdot \delta(x)$
- ▶ Gauss($x; \mu, \nu$) = $[\nu = 0] \cdot \delta(x - \mu) + [\nu \neq 0] \cdot \frac{e^{-(x-\mu)^2/(2\nu)}}{(2\pi\nu)^{\frac{1}{2}}}$
- ▶ UniformInt($x; a, b$) = $\frac{\sum_{x' \in \mathbb{Z}} \delta(x - x') \cdot [a \leq x'] \cdot [x' \leq b]}{\sum_{x' \in \mathbb{Z}} [a \leq x'] \cdot [x' \leq b]}$

Symbolic Simplification

- ▶ Basic algebraic simplifications

$$x + x \rightarrow 2 \cdot x, \quad x \cdot x \rightarrow x^2, \dots$$

- ▶ Simplifications on constraints

$$[x = 0] + [x \neq 0] \rightarrow 1, \quad [x \leq 0] \cdot [0 \leq x] \rightarrow [x = 0],$$

$$\delta(x) \cdot [1 \leq x] \rightarrow 0, \dots$$

- ▶ Linearize Guards

$$\delta(y - x^2) \rightarrow$$

$$[-y \leq 0] \cdot ([x = 0] \cdot \delta(y) + [x \neq 0] \cdot \frac{1}{2\sqrt{y}}(\delta(x - \sqrt{y}) + \delta(x + \sqrt{y}))),$$

...

- ▶ Symbolic Integration

$$\begin{aligned}
 \int_{\mathbb{R}} dx \int_{\mathbb{R}} dy [0 \leq x] \cdot [x \leq 1] \cdot [0 \leq y] \cdot [y \leq 1] \cdot \delta(z - x \cdot y) \rightarrow \\
 -[0 < z] \cdot [z \leq 1] \cdot \log(z), \dots
 \end{aligned}$$

Learning-based Programming Engines

SLANG

JS NICE

NICE 2 Predict

DEGUARD

Deep3

1. **Pick** a structure of interest, e.g., trees:



2. **Define** a DSL for expressing functions:
(can be Turing complete)

```
TCond ::= z | WriteOp TCond | MoveOp TCond
MoveOp ::= Up, Left, Right, DownFirst, DownLast,
           NextDFS, PrevDFS, NextLeaf, PrevLeaf,
           PrevNodeType, PrevNodeValue,
WriteOp ::= WriteValue, WriteType, WritePos
```

3. **Synthesize** $f_{best} \in \text{DSL}$ from Dataset D:

$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}} \text{cost}(D, f)$$

4. **Use** f_{best} on new structures:

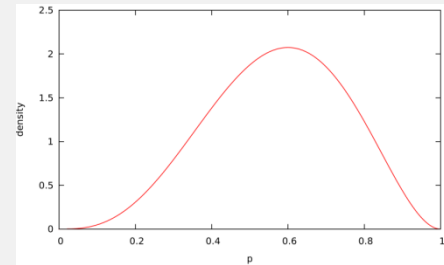
$$f_{best} (\text{tree}) \rightarrow \gamma$$

<http://plml.ethz.ch>

PSI: Probabilistic Solver

PSI SOLVER

```
def main() {
  p := Uniform(0,1);
  r := [1,1,0,1,0];
  for i in [0..r.len] {
    observe(
      Bernoulli(p) == r[i]);
  }
  return p;
}
```



<http://psisolver.org>